

程序培训 VEXCODE V5 PRO

1. 下载和安装



下载地址: [VEXcode Install V5 - VEX Robotics](#)

Download VEXcode V5 (Blocks and Text) - v2.4.5

Downloadable app for users with bandwidth restrictions and for convenience

Download for
Windows

Download for
Mac

Available in the
Chrome Web Store

Download on the
App Store

MSI (For IT)

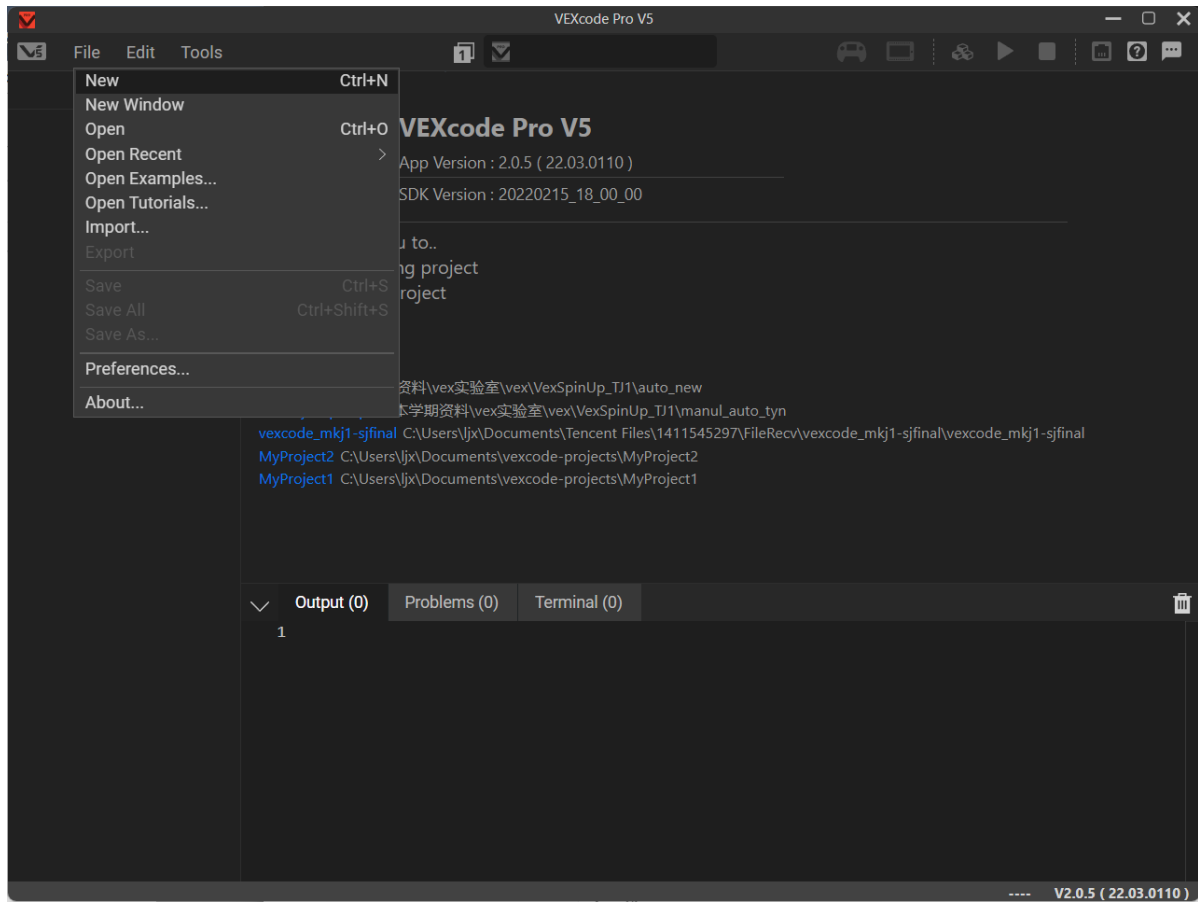
MSI Help

GET IT ON
Google Play

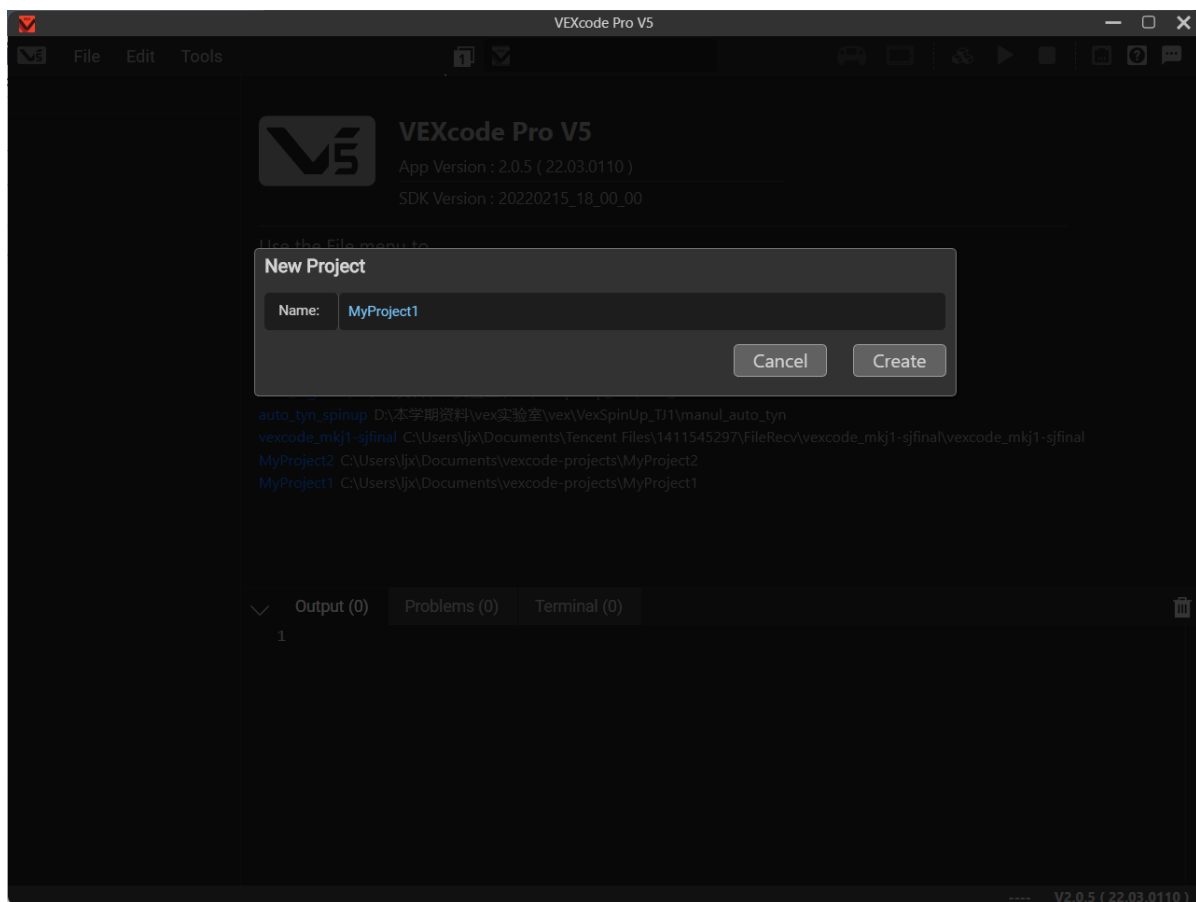
available at
amazon appstore

2. 基础使用

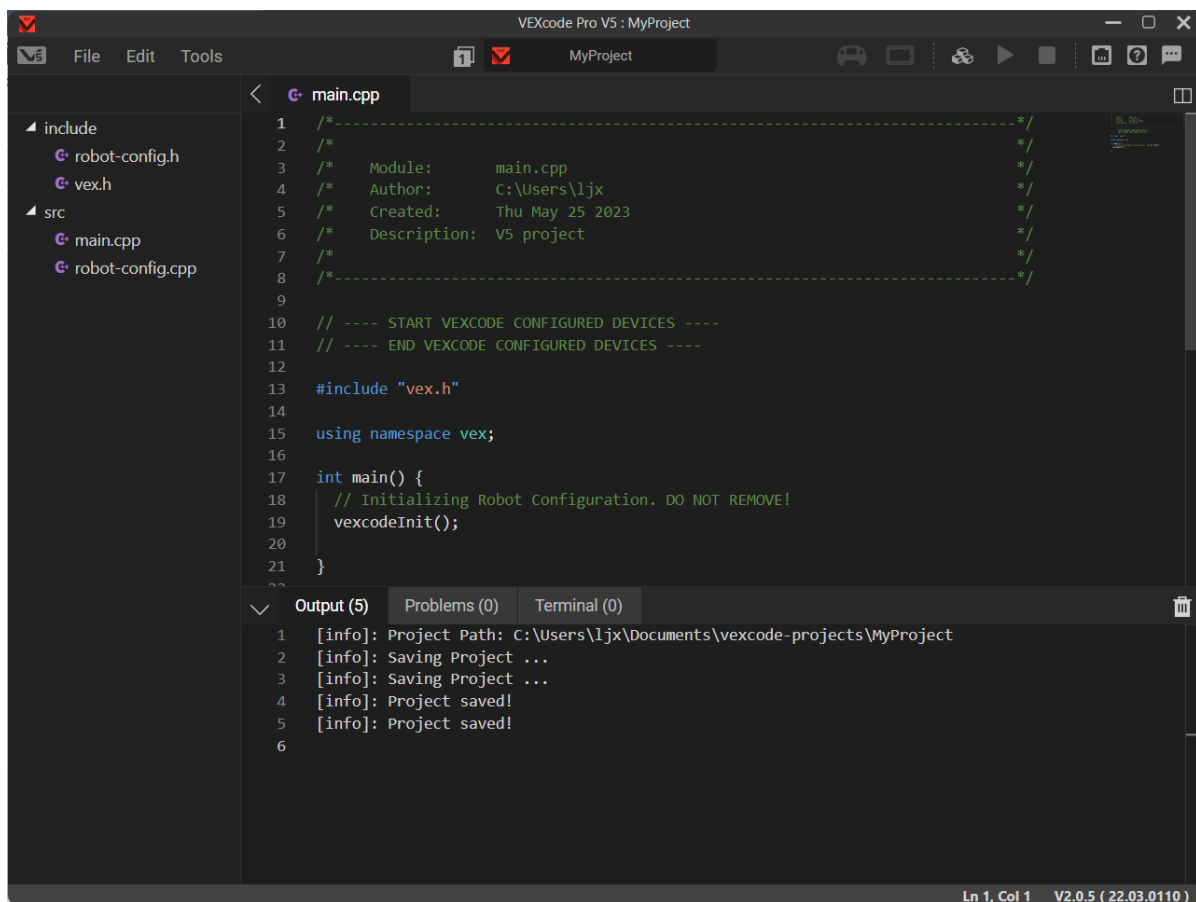
2.1 新建项目



点击File -> New. (或Ctrl + N)



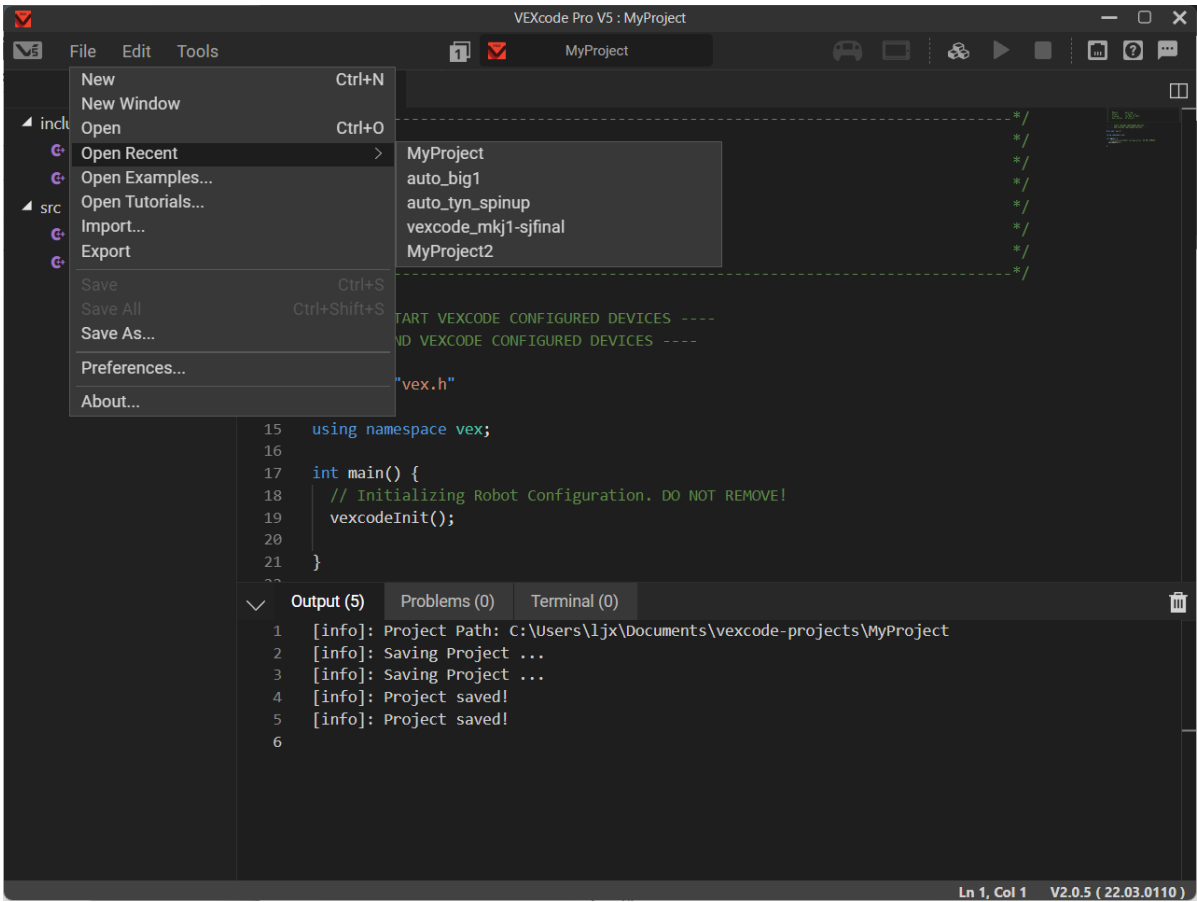
点击create



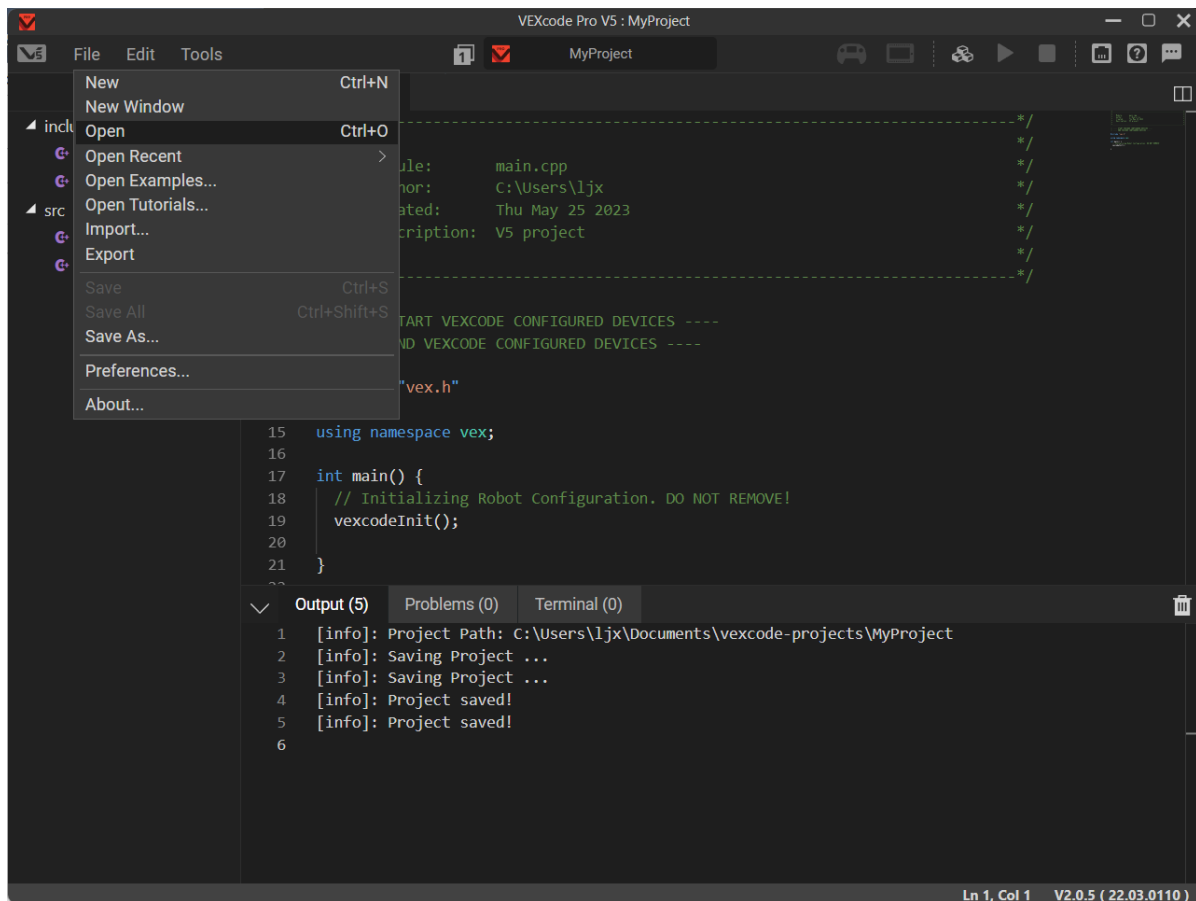
创建完成!

2.2 打开项目

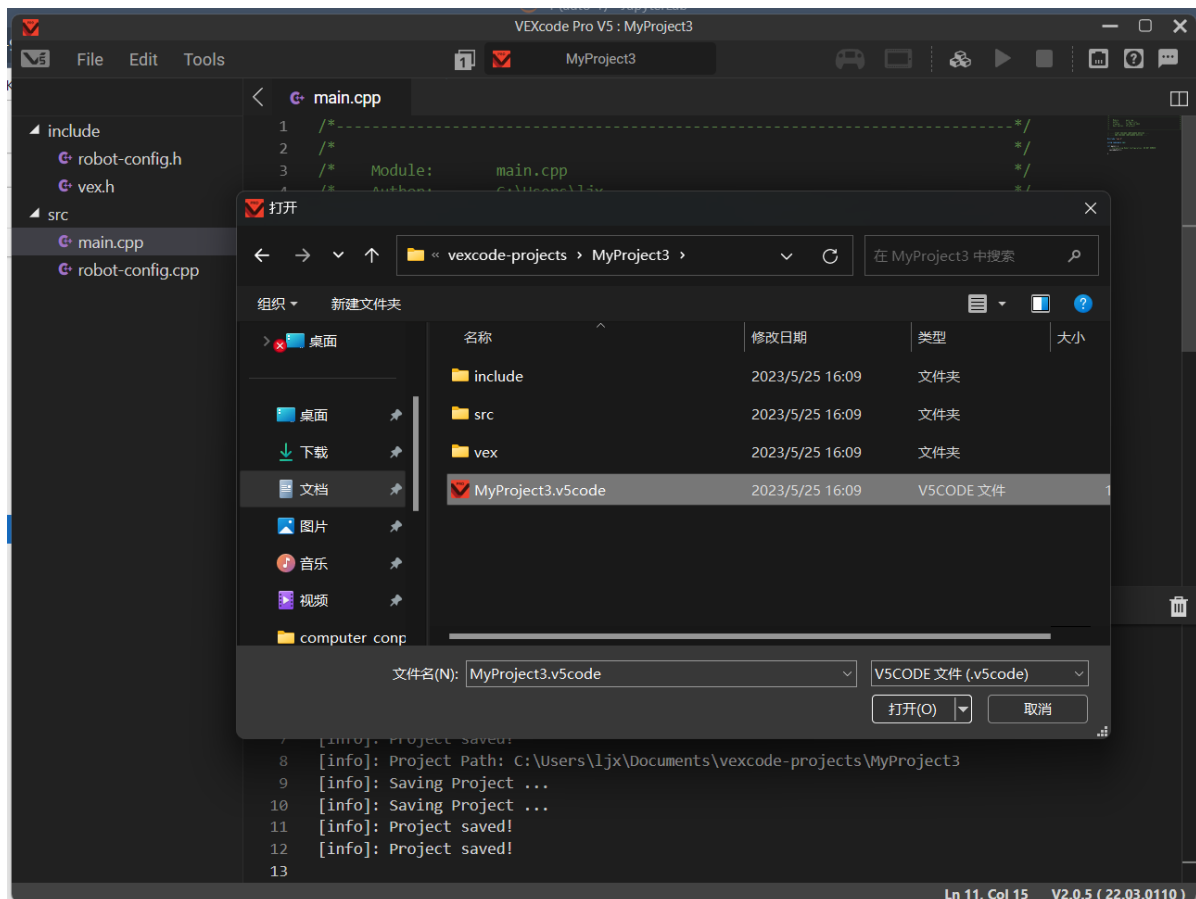
2.2.1 方法1: Open Recent



2.2.2 方法2: 打开对应项目的.v5code文件

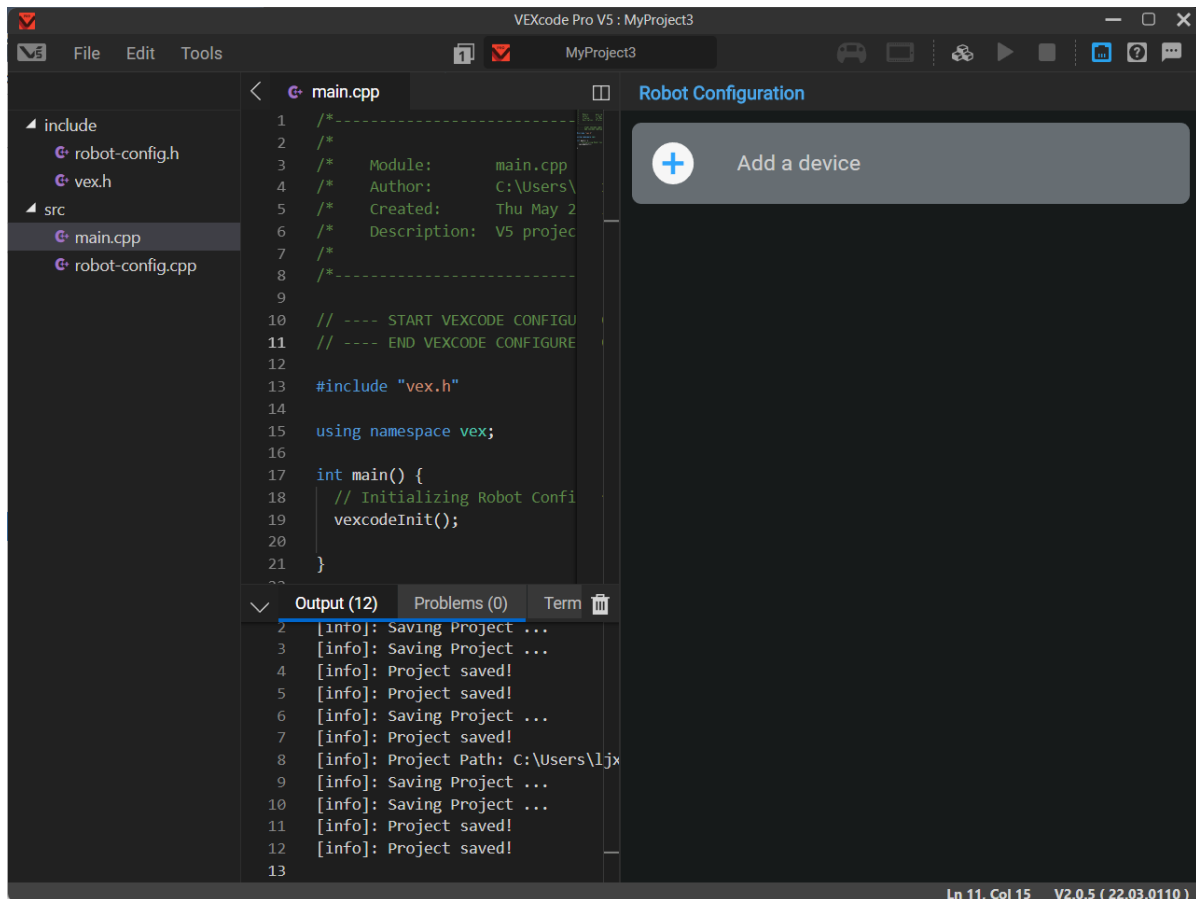


点击File -> Open (Ctrl + O)

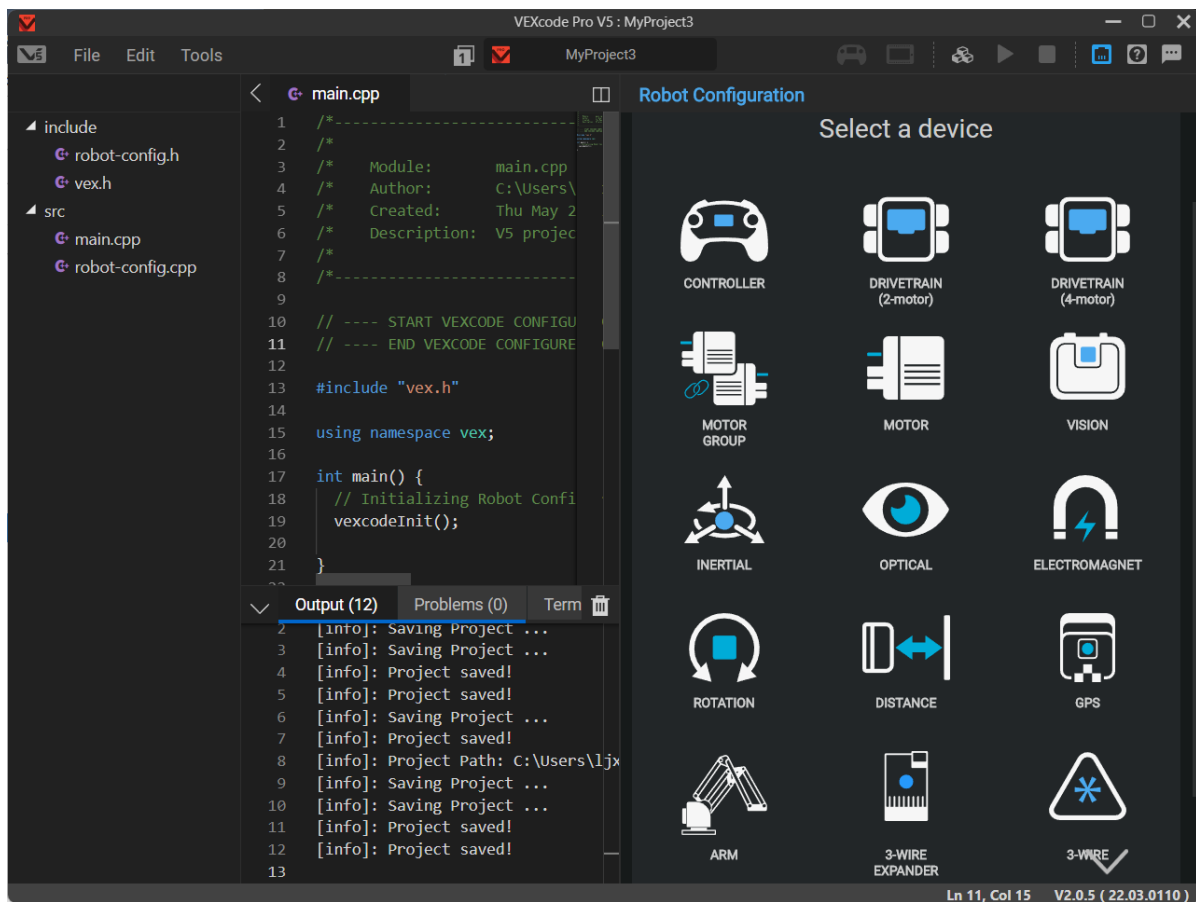


.v5code类似于visual studio项目中的.sln文件, 点击打开, 打开对应的项目

2.3 定义端口

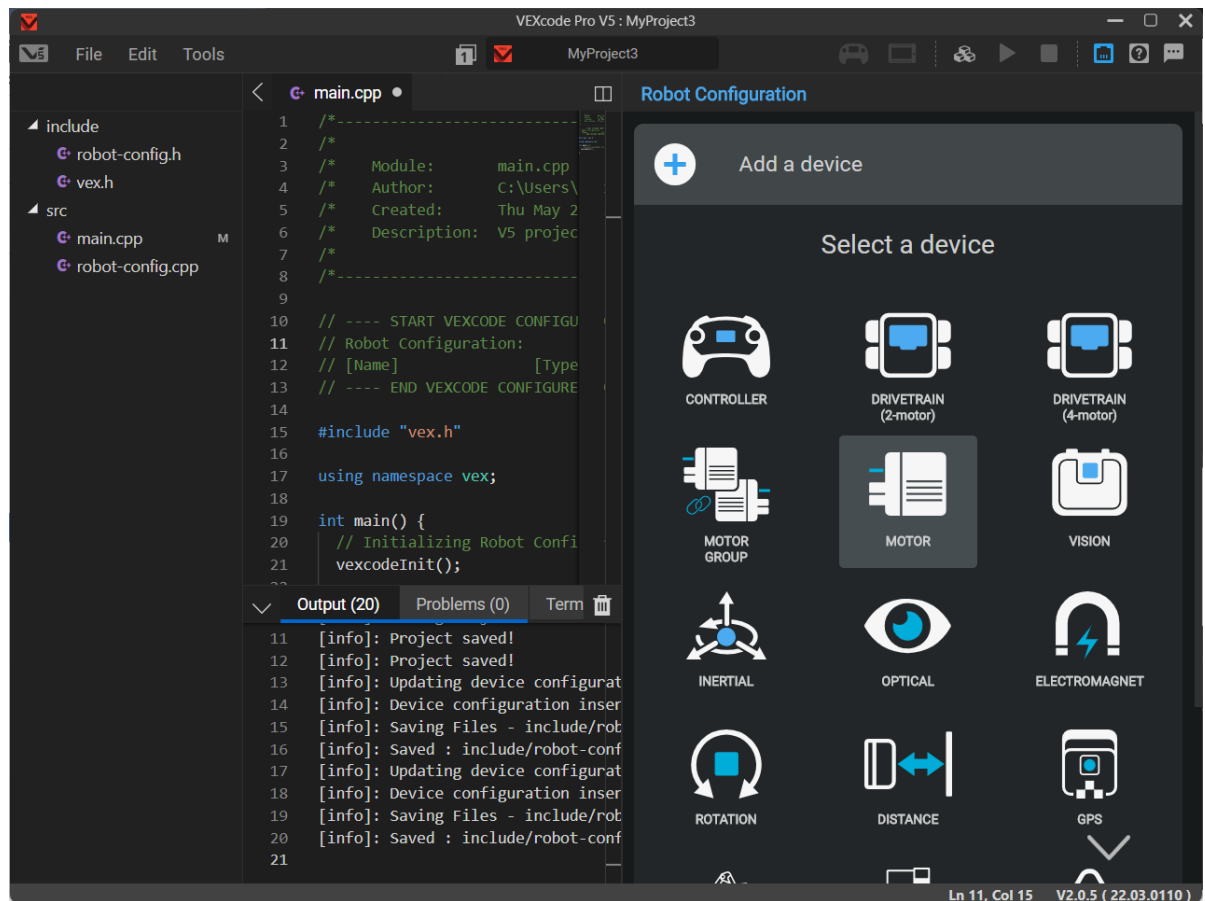


点击右上角端口图标 -> 点击Add a Device

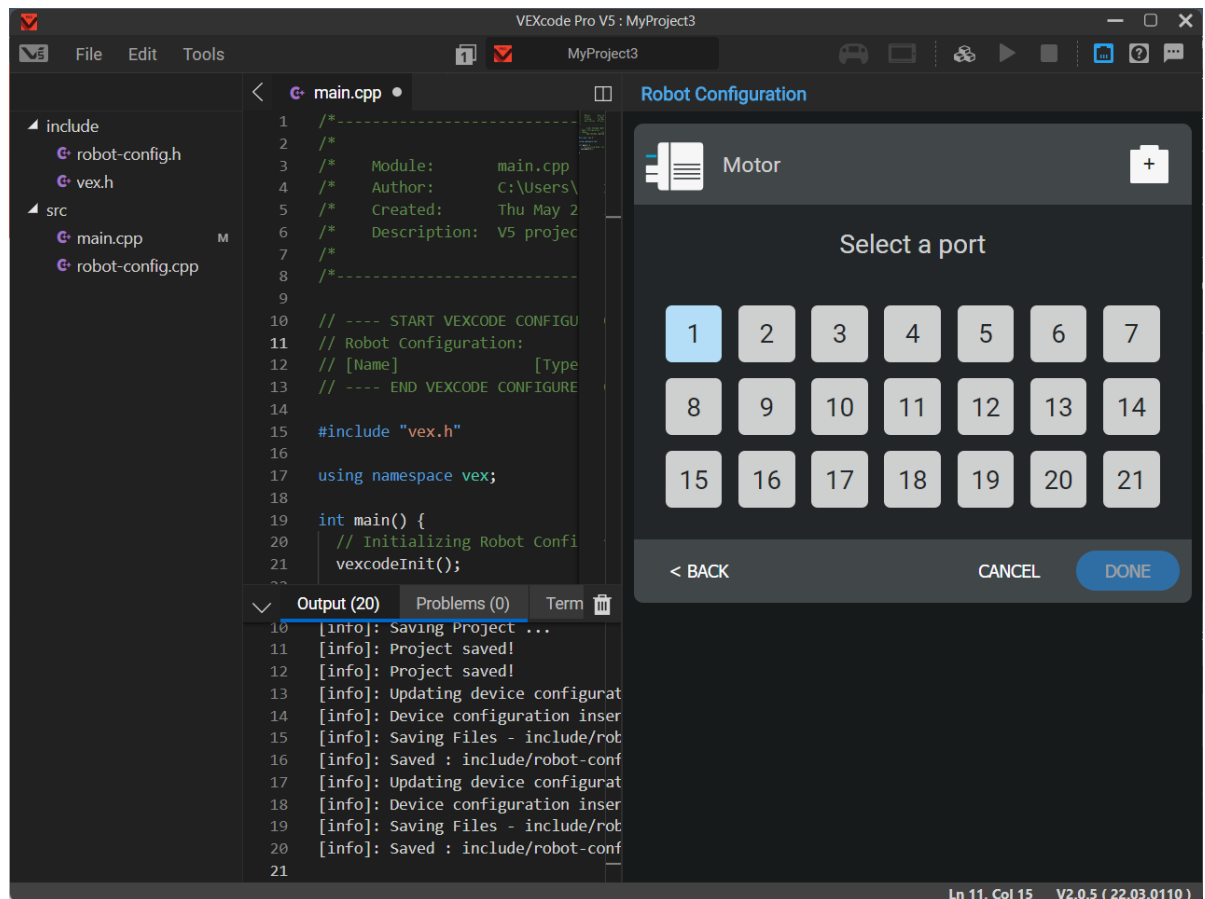


有多种设备供大家选择，下面以一个底盘需要的设备为例

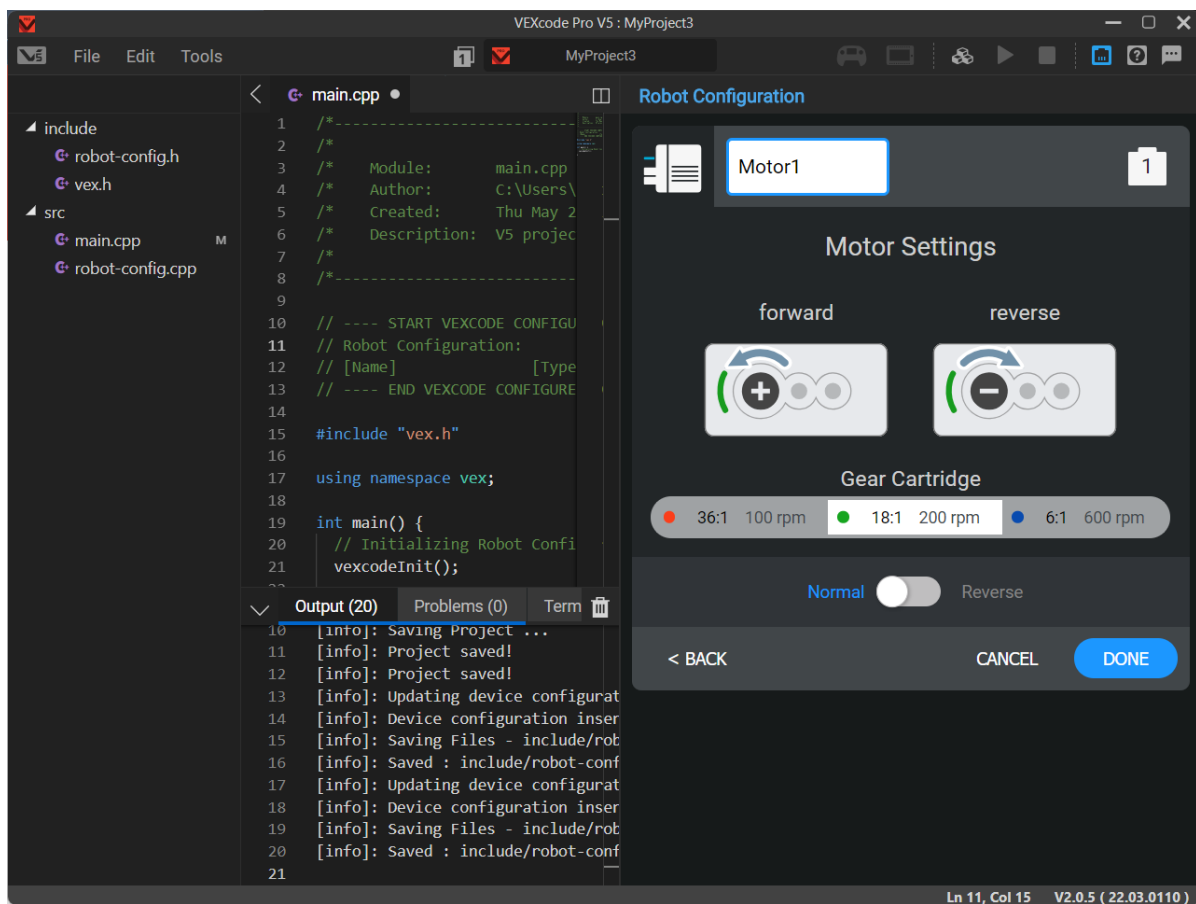
2.3.1 定义电机



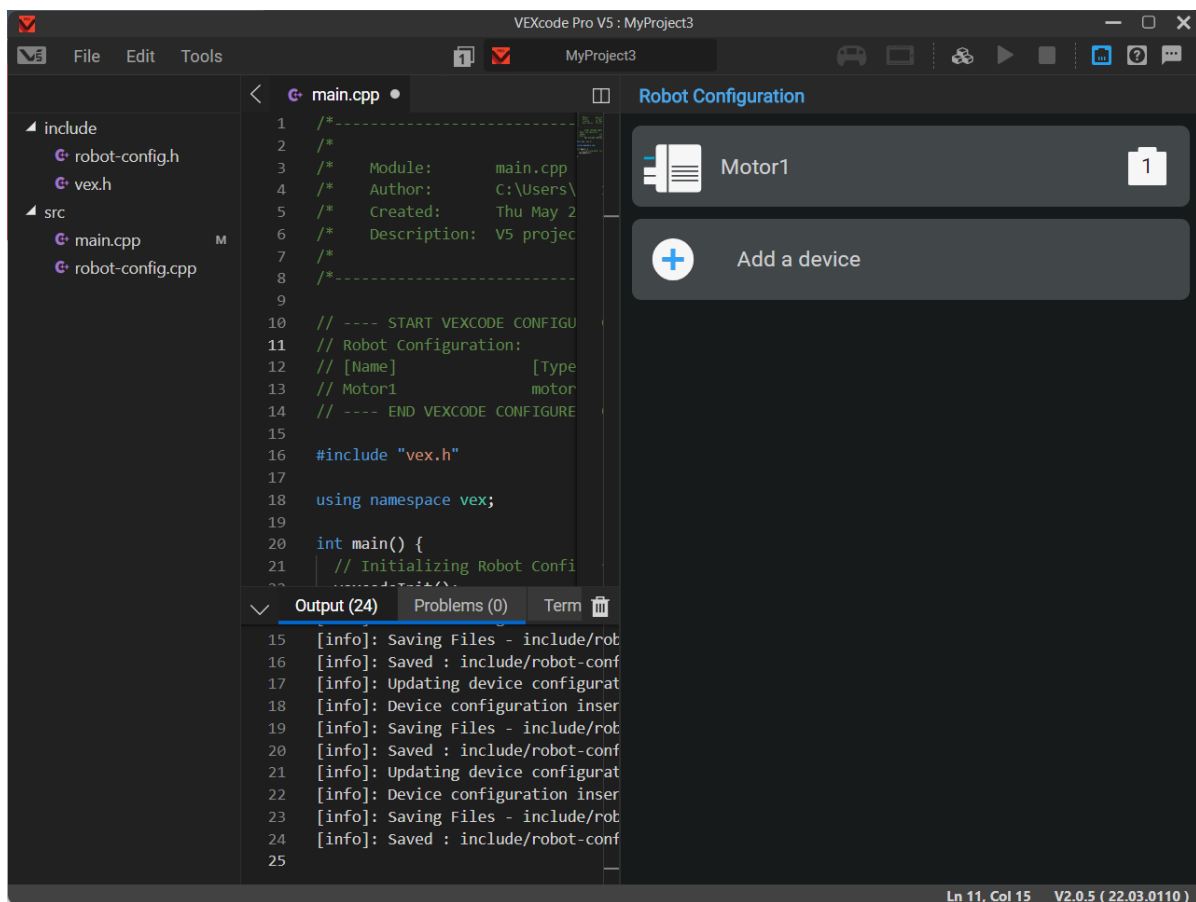
点击电机



选择电机对应的连接在主控上的端口



选择相应的电机类型（红绿蓝），选择正反转，完成后点击DONE




```
#include "vex.h"

using namespace vex;
using signature = vision::signature;
using code = vision::code;

// A global instance of brain used for printing to the V5 Brain screen
brain Brain;

// VEXcode device constructors
motor Motor1 = motor(PORT1, ratio18_1, false);

// VEXcode generated functions

/**
 * Used to initialize code/tasks/devices added using tools in VEXcode Pro.
 */
```

Output (24) Problems (0) Terminal (0)

```
[info]: Device configuration inserted few comments in the file: src/main.cpp
[info]: Saving Files - include/robot-config.h, src/robot-config.cpp
[info]: Saved : include/robot-config.h, src/robot-config.cpp
[info]: Updating device configuration...
[info]: Device configuration inserted few comments in the file: src/main.cpp
[info]: Saving Files - include/robot-config.h, src/robot-config.cpp
[info]: Saved : include/robot-config.h, src/robot-config.cpp
[info]: Updating device configuration...
[info]: Device configuration inserted few comments in the file: src/main.cpp
[info]: Saving Files - include/robot-config.h, src/robot-config.cpp
[info]: Saved : include/robot-config.h, src/robot-config.cpp
```

```
using namespace vex;

extern brain Brain;

// VEXcode devices
extern motor Motor1;

/**
 * Used to initialize code/tasks/devices added using tools in VEXcode Pro.
 * This should be called at the start of your int main function.
 */
void vexcodeInit( void );
```

Output (24) Problems (0) Terminal (0)

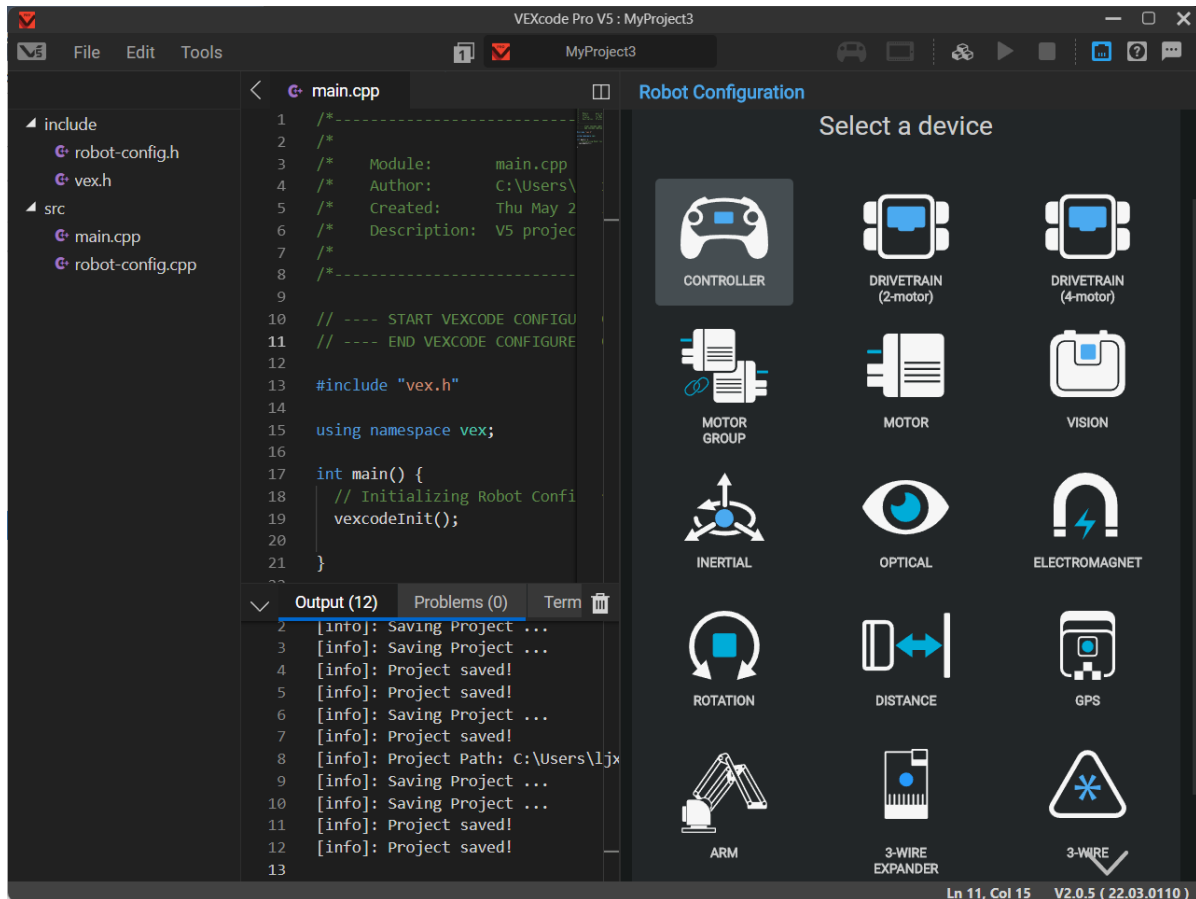
```
[info]: Device configuration inserted few comments in the file: src/main.cpp
[info]: Saving Files - include/robot-config.h, src/robot-config.cpp
[info]: Saved : include/robot-config.h, src/robot-config.cpp
[info]: Updating device configuration...
[info]: Device configuration inserted few comments in the file: src/main.cpp
[info]: Saving Files - include/robot-config.h, src/robot-config.cpp
[info]: Saved : include/robot-config.h, src/robot-config.cpp
[info]: Updating device configuration...
[info]: Device configuration inserted few comments in the file: src/main.cpp
[info]: Saving Files - include/robot-config.h, src/robot-config.cpp
[info]: Saved : include/robot-config.h, src/robot-config.cpp
```

完成定义。可以看到相应的文件中也加入了相关的定义。

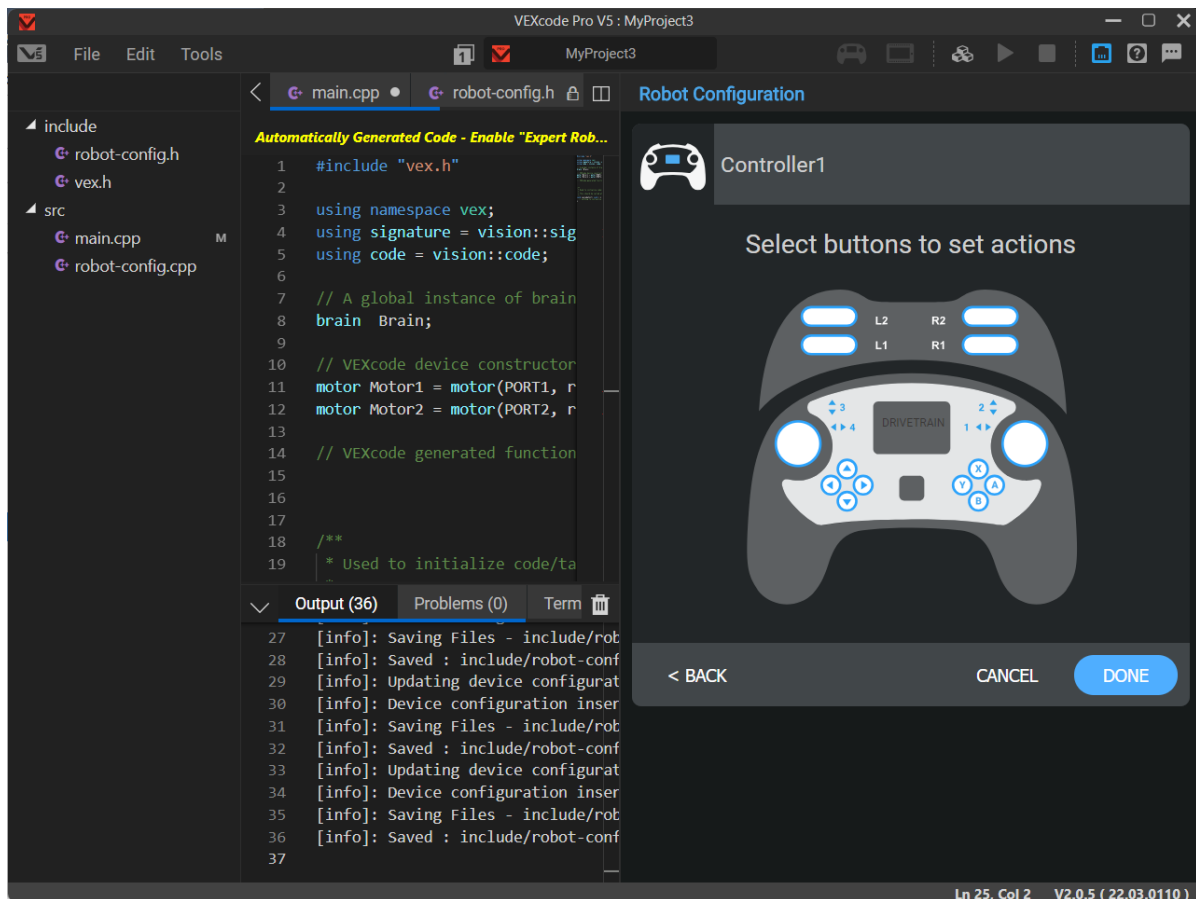
也可以修改robot-config.cpp和robot-config.h的内容定义相应的端口，但在VEXcode Pro V5中会产生问题，**本次校内赛中不建议使用**，今后使用vscode+vexcode pro或pros插件时使用修改文件方式定义。

按上述步骤定义其他电机即可。

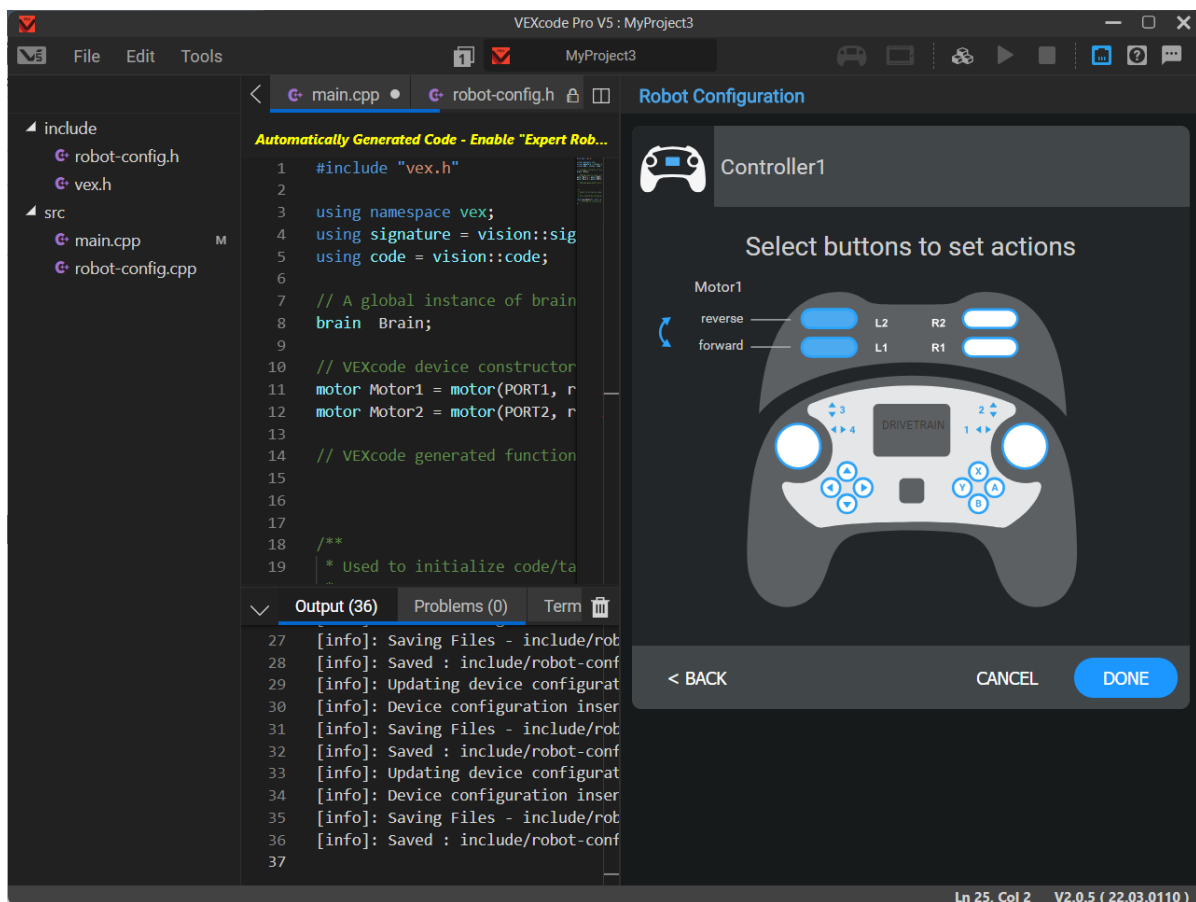
2.3.2 定义Controller



点击Controller



直接点击DONE



注：点击相应按键会出现vex自动定义的相关操作，但此方法定义后不方便修改，不够灵活，故不推荐使用

2.3.3 其他设备

其他设备请参照程序的提示以及网上相关资料自行进行学习

3. 项目基础说明

3.1 头文件和命名空间说明

```
< main.cpp •  
  
220  
221 #include "vex.h"  
222  
223  
224 using namespace vex;
```

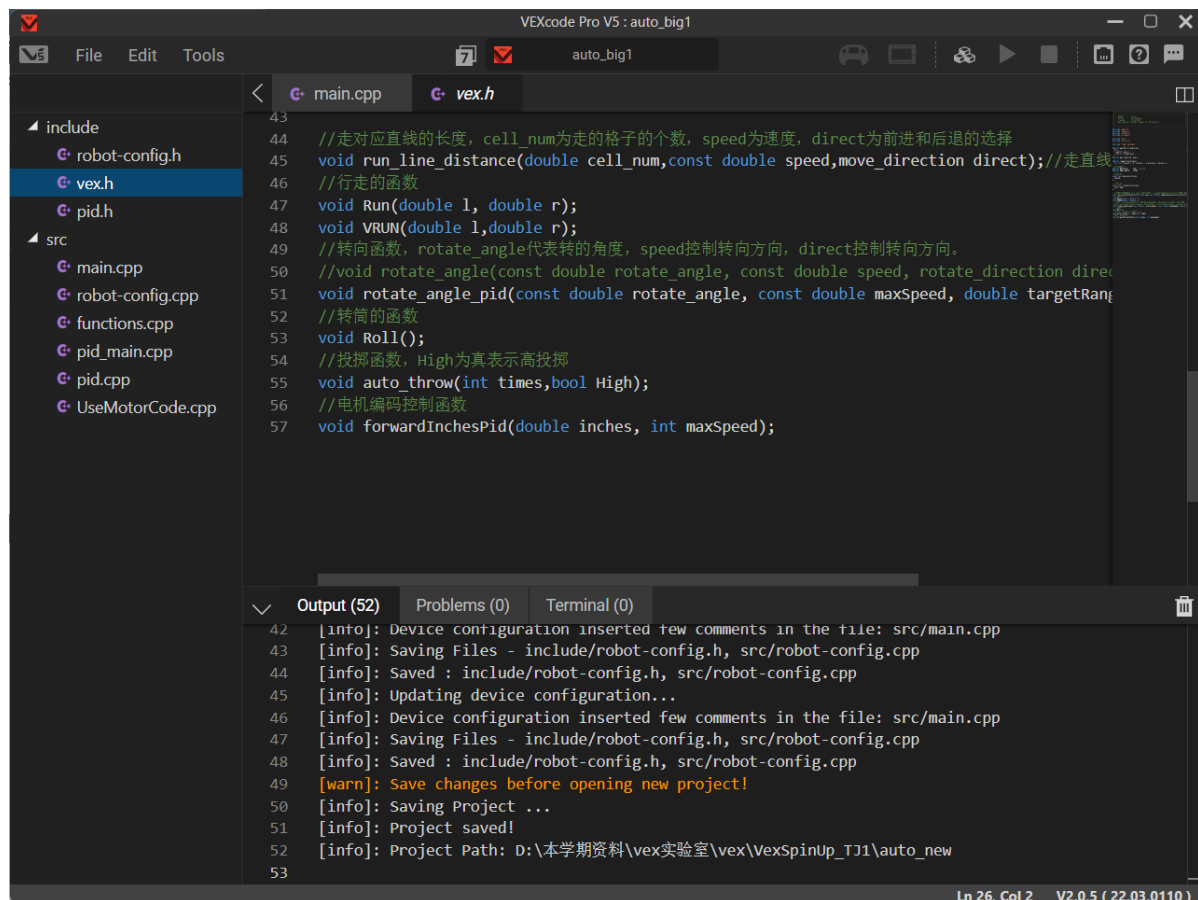
项目文件中需包含vex.h，同时命名空间是vex而不是std。这会导致一些标准库中的函数需要加上命名空间前缀才能使用，如一些数学函数max()需写为std::max()。

```
1 #include "vex.h"
2 #include "robot-config.h"
3
4 using namespace vex;
```

3.2 项目结构

程序结构可自行组织，如果不想定义相关函数在头文件的声明可以将函数都定义在main.cpp中。

如果多文件，可以参照高程中多文件定义方式，将函数声明放在vex.h中（当然其他头文件也是可以）。在相应的cpp中引用相应的头文件。以下为例



4. 手动程序

4.1 基础手动程序

4.1.1 电机类的一些代码

下面代码中假设已经经过上述步骤定义完成电机**Motor_test**，以Motor_test来说明相应的成员函数

```
1 namespace vex{
2 class vex::motor{
3     //非开源代码，仅仅说明作用
4
5     /*****按照一定的速度或电压转动*****/
6     /**
7      * @brief 打开马达并以指定的方向和速度旋转。
8      * @param dir 旋转电机的方向。
9      * @param velocity 设置速度的大小。
10     * @param units 速度值的测量单位。
11     */
12     void spin( directionType dir, double velocity, velocityUnits units );
13
14     void spin( directionType dir, double velocity, percentUnits units );
15     //第一个为速度单位，有rpm，第二个为百分比单位，有pct，具体看api
16
17     /**
18      * @brief 开启电机，以指定的方向和指定的电压旋转。
19      * @param dir 旋转电机的方向。
20      * @param voltage 设置电压的大小。
21      * @param units 电压值的单位。
22      */
23     void spin( directionType dir, double voltage, voltageUnits units );
24
25
26     /*****转动一定的角度*****/
27     /**
28      * @brief 开启电机，并以指定的速度将其旋转到一个相对的目标旋转值。
29      * @return 返回一个布尔值，标志着电机已经达到目标旋转值。
30      * @param rotation 设置旋转的数量。
31      * @param units 旋转值的测量单位。
32      * @param velocity 设置速度的数量。
33      * @param units_v 速度值的测量单位。
34      * @param waitForCompletion (Optional) 如果是true，你的程序将等待，直到马达达到
      目标旋转值。如果为假，程序将在调用此函数后继续进行。默认情况下，这个参数为真。
35      */
36     bool spinFor( double rotation, rotationUnits units, double velocity,
      velocityUnits units_v, bool waitForCompletion=true );
37
38     bool spinFor( directionType dir, double rotation, rotationUnits units,
      double velocity, velocityUnits units_v, bool waitForCompletion=true );
39     //spinFor还有其他重载函数，具体看api
40
41     /*****直接转动(不常用)*****/
42     /**
43      * @brief 打开马达，并按指定的方向旋转。
44      * @param dir 电机旋转的方向。
45      */
46     void spin( directionType dir );
47     //需配合相关的set来设置基础参数，基本不用
48 }
```

```

49  /*****set相关*****/
50  /**
51  *@brief 通过传递一个制动模式作为参数来设置电机的停止模式。
52  *@param mode 停止模式可以被设置为coast, brake, or hold.
53  */
54  void setStopping( brakeType mode );
55  //其他看api
56  };
57

```

4.2 底盘移动的基础函数

底盘移动提供两个基础的函数，分别是带pid用spin来写的Run() 以及 不带pid的VRUN() (推荐)

```

1  //均为4电机示例，其他电机数进行相应的修改
2  //前进函数1
3  //假设已经定义好了对应的电机
4  void Run(double l, double r) {
5      FrontLeft.spin(fwd, l, pct);
6      BackLeft.spin(fwd, l, pct);
7      FrontRight.spin(fwd, r, pct);
8      BackRight.spin(fwd, r, pct);
9  }
10
11 //前进函数2
12 void VRUN(double l, double r)
13 {
14     vexMotorVoltageSet(vex::PORT1, l*120);           //PORT为电机对应的端口
15     //vexMotorVoltageSet(FrontLeft.index(), l*120); //也可以使用已经定义电机的引
16     索
17     vexMotorVoltageSet(vex::PORT2, l*120);
18     vexMotorVoltageSet(vex::PORT3, r*120);
19     vexMotorVoltageSet(vex::PORT4, r*120);
20 }

```

4.3 遥杆控制写法

遥感控制代码写在main函数中

```

1  #include <cmath>
2  int main() {
3      vexcodeInit();
4
5      while(1)
6      {
7          /**操纵**/
8          int fb, lf;
9          /*****
10         相应Axis对应(两个十字对应手柄左右两边遥感，可能有误):

```

```

11         Axis1
12         =
13         Axis2 =====
14         =             Axis3
15                         =
16                         Axis4===
17                         =
18         *****/
19         fb=Controller1.Axis3.value();
20         lf=Controller1.Axis4.value();
21         fb=std::abs(fb)>15?fb:0;
22         lf=std::abs(lf)>15?lf:0;
23         if(fb!=0||lf!=0) VRUN((fb+lf)*100.0/127.0,(fb-lf)*100.0/127.0);//此处
也可用Run(), 具体差别自行用机器人体会
24         else Run(0,0);
25
26
27
28
29         //...其他操作
30
31
32
33
34         sleep(8);//注意要sleep一小段时间防止过载
35     }
36
37 }

```

4.4 其他功能写法（示例）

手动中其他功能写在main函数中的while循环内，下面以SpinUp中转动投盘电机为例

```

1  while(1){
2      //底盘遥感控制，同上
3
4      //提前定义好了投盘电机ShootMotor
5      //按Y键转动，松开停止
6      if(Controller1.ButtonR1.pressing()){
7          //相应功能根据具体操作进行替换，此处为转动电机
8          ShootMotor.spin(forward,50,pct);
9      }
10     else{
11         //注意不按R1的时候需要让电机停止转动
12         ShootMotor.spin(forward,0,pct);
13     }
14
15
16     //...
17
18     //再次提醒注意延时

```

```

19     sleep(8);
20
21 }

```

如果需要实现的功能较为复杂，可以定义函数，在main中引用相应的函数。

有的功能（比如想要自己写电机的pid控制，需要用到线程）

4.5 整体示例

一个可以用遥感控制底盘，带有飞盘发射功能的手动代码如下

```

1  #include <cmath>
2  int main() {
3      vexcodeInit();
4
5      while(1)
6      {
7          /**操纵**/
8          int fb,lf;
9          /*******
10         相应Axis对应(两个十字对应手柄左右两边遥感，可能有误):
11             Axis1
12                 =
13         Axis2 =====
14                 =             Axis3
15                             =
16                             Axis4===
17                             =
18         *****/
19         fb=Controller1.Axis3.value();
20         lf=Controller1.Axis4.value();
21         fb=std::abs(fb)>15?fb:0;
22         lf=std::abs(lf)>15?lf:0;
23         if(fb!=0||lf!=0) VRUN((fb+lf)*100.0/127.0,(fb-lf)*100.0/127.0);//此处
也可用Run(), 具体差别自行用机器人体会
24         else Run(0,0);
25
26         //提前定义好了投盘电机ShootMotor
27         //按Y键转动，松开停止
28         if(Controller1.ButtonR1.pressing()){
29             //相应功能根据具体操作进行替换，此处为转动电机
30             ShootMotor.spin(forward,50,pct);
31         }
32         else{
33             //注意不按R1的时候需要让电机停止转动
34             ShootMotor.spin(forward,0,pct);
35         }
36
37
38         //控制的推盘电机
39         if(Controller1.ButtonA.pressing()){
40             pushPan.spinFor(reverse,360,degrees,50,
(velocityUnits)pct,false);

```



```

41         sleep(400); //延时随意
42     }
43
44
45
46     sleep(8); //注意要sleep一小段时间防止过载
47 }
48
49 }
```

机器人结构如下图：

5. 自动程序

5.1 自动控制的函数

5.1.1 通过Run()或VRun()控制

在自动程序中，通过设置Run()或VRun()的延时来控制相应的距离，函数实现较容易，但非常难以调试。具体使用：

```

1 Run(-30,-30);
2 Sleep(300); //时间自行调试
3 Run(0,0);
```

5.1.2 通过记录一个固定距离的时间来进行封装

通过测试出走一段固定距离（如一个格子，即2 inch）的延时，写一个移动的函数

其中一个示例：

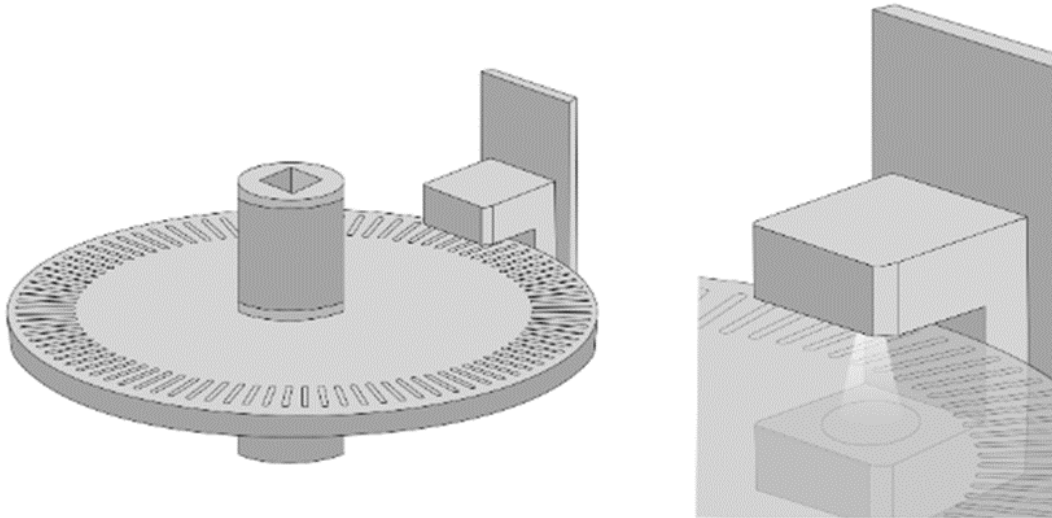
```

1 void run_line_distance(double cell_num,double speed)
2 {
3     //RUN_LINE_K自行测试
4     const double K = RUN_LINE_K; //对应常数，可以进行调试得到最终的长度
5     double run_time=cell_num/speed*K;
6     speed *= (direct==move_direction::fwd?1:-1);
7     printf("into auto");
8     Run(speed,speed);
9     task::sleep(run_time);
10    Run(0,0);
11 }
```

此方法本质与法1相同，也不是很准确，不过可以稍微方便调参

5.1.3 电机编码控制

培训中提到了编码器，如下图：



电机中自带编码器，可以获得电机转动的总角度。可通过此数据来编写相应的移动函数。

给出一个示例框架，不过无法直接运行，若要使用需大家自行补齐

```
1  /*
2      LFA,LBA,RFA,RBA为已经定义的电机
3  */
4  const double TRACKING_CIRCUMFERENCE = 4 * PI; // 跟踪轮的周长（直径*PI）//还需要
    测量
5  double change = 4/3; // 改变距离，因为齿轮比
6
7  // 使用尺寸分析法在跟踪轮的英寸和刻度之间进行转换
8  double inchesToTicks(double inches)
9  {
10     return inches * (360 / TRACKING_CIRCUMFERENCE) * change;
11 }
12
13 double ticksToInches(double ticks)
14 {
15     return ticks * (TRACKING_CIRCUMFERENCE / 360) * change;
16 }
17
18 //停止
19 void stopBase()
20 {
21     LFA.stop(coast);
22     LBA.stop(coast);
23
24     RFA.stop(coast);
25     RBA.stop(coast);
26 }
27
28 double getTotalDistance() // return average motor encoder value
```

```

29 {
30     // skip L2 cuz thats the roller/intake and R2 cuz thats the cata
31
32     return ticksToInches(
33         LFA.position(deg) +
34         RFA.position(deg)) / 2;
35 }
36
37 //返回夹在两个数字之间的数字
38 double keepInRange(double n, double bottom, double top) {
39     if (n < bottom)
40         n = bottom;
41     if (n > top)
42         n = top;
43     return n;
44 }
45
46
47 // 逐渐加速和减速
48 // 用负距离来倒退
49 // More info: https://www.vexforum.com/t/advanced-pid-and-motion-profile-control/28400/3
50 //以最大速度为maxSpeed前进inches
51 void forwardInches(double inches, int maxSpeed)
52 {
53     resetTotalDistance();
54     //maxSpeed = keepInRange(maxSpeed, 0, 100);
55
56     //参数自己调
57     //MinSpeed : 确保运动克服了摩擦
58     const double minSpeed = 3;          // 电机的最低速度；低速时转弯更精确，但也更笨重。
59
60     // 加速率：改变加速/减速的速度
61     // 前半段逐渐加快，后半段逐渐减慢
62     const double accelRate = 100;      // 加速时的速度倍率（开始时的斜率）。
63     const double deaccelRate = 4;      // 减速时的速度倍率（末尾的斜率）。较高时开始放慢
64     速度
65
66     double targetDistance = inches; /*inchesToTicks(inches)*;/ // 车应该走多远
67     double targetRange = .25;          // 与车必须停止的理想距离的
68     距离。
69     double error = targetDistance;      // 与预期距离的距离
70     double speed;                       // 电机的实际速度值
71
72     while (fabs(error) > targetRange) {
73
74         error = targetDistance - getTotalDistance(); // 误差=期望值-实际值
75
76         // 前半段：加速到最大。后半段，减速到最小
77         // 速度与误差/目标距离成正比。
78         // 由于误差总是在减少，所以速度比例在前一半的距离中是倒置的（从1中减去）。
79         // 走过的路程，所以速度是按比例增加而不是减少的。
80         // fabs()（绝对值）被使用，因为方向是由误差的符号决定的，而不是由速度决定的。
81         if (fabs(error) > fabs(targetDistance/2))
82         {

```

```

81     speed = (1 - (fabs(error) / fabs(targetDistance))) * accelRate; //随着
误差的减少，速度加快
82     }
83     else
84     {
85         speed = (fabs(error) / fabs(targetDistance)) * deaccelRate; // 随着误
差的增加而减慢速度
86     }
87     speed = fabs(speed);
88     speed *= maxSpeed;
89     speed = keepInRange(speed, minSpeed, maxSpeed);
90
91     // 如果有正的误差就向前走，如果有负的误差就向后走
92     if (error < 0)
93     {
94         speed *= -1;
95     }
96     VRun(speed, speed)
97
98     //在主控上打印相应的信息
99     Brain.Screen.printAt(1, 60, "Target Dist: %.2f ", (targetDistance));
100    Brain.Screen.printAt(1, 80, "Progress:   %.2f ",
(getTotalDistance()));
101    Brain.Screen.printAt(1, 120, "Error:      %.2f      ", error);
102    Brain.Screen.printAt(1, 100, "Speed:     %.2f   ", speed);
103    }
104
105    stopBase();
106    task::sleep(15);
107 }

```

5.1.4 pid控制

pid控制可以根据调整参数达到想要的控制效果，如果有时间，请自行查阅资料，在自动阶段的底盘移动，或其他你觉得需要加入pid控制的地方加入pid控制，鼓励大家这么做。

pid的资料请在github或vexforum等网站自行查找，文档不再给出。

6. 线程的使用和定义

6.1 线程的概念

由于vex采用C++和Python，其运行顺序按照指令逐步运行，因此如果想并行执行几项操作（比如边移动边用pid控制转动的飞轮），则需要用到线程。

6.2 线程的定义

线程定义用到了vex::task类，其构造函数如下：

```
1  /**
2   * @brief 构建一个带有函数回调的任务。
3   * @param callback 对一个函数的引用。
4   */
5   task( int (* callback)(void) );
6
7  /**
8   * @brief 构建一个带有函数回调的任务。
9   * @param callback 对一个函数的引用。
10  * @param arg 一个空指针，被传递给回调。
11  */
12  task( int (* callback)(void *), void *arg );
```

还有其他一些带优先集的定义方法，如有需要自行研究（不过新生赛应该用不到这么复杂）

对第一个定义方法，给出以下示例：

```
1  int opr(){
2      //...
3
4      //task如果要一直进行需要写while(1)
5      while(1){
6          //...
7      }
8      return 0; //注意要return一个值
9  }
10
11  vex::task(opr);
```

如果需要传参，请参照以下方式

```
1  struct args{
2      double args1;
3      //...自行定义相关参数
4  };
5
6  int test(void * args_1)
7  {
8      args* fargs = (args*) args_1;
9      //相关操作
10     //...
11     while(1){
12         //...
13     }
14     return 0;
15 }
16
17 int main(){
18     args args_1{...}; //相应初始化
```

```
19 task t1(test, (void*) &args_1);
20 }
```

6.3 线程的操作

在线程定义后，线程将自动启动。如果是暂停的线程，通过resume()方法来继续。

线程的停止有suspend()和stop()两种方法，其中suspend()是暂停线程，线程还可以接着使用。而stop()是中止线程，线程无法再使用resume()方法恢复。

使用示例

```
1 using namespace vex;
2 //假设定义了test为功能模块函数
3 int auto(){
4     task t1(test); //t1开始运行
5     t1.suspend(); //t1暂停运行
6
7     //...
8
9     t1.resume(); //t1继续运行
10    //...
11    //在resume到这里t1都会运行
12
13    t1.stop(); //t1停止
14    t1.resume(); //此句为无效语句。（好像会报错好像也不会，但注意stop之后就不要调用了）
15 }
```

7. 参考网站

vex论坛: [VEX Forum - A forum to discuss VEX Robotics.](https://forum.vexrobotics.com/)

vexcode pro V5 api: [VEX Help \(vexcode.cloud\)](https://vexcode.cloud/)

github: <https://github.com>

如需要梯子，可用: <https://ikuuu.uk/> (不要说是我发的)